StarPU Handbook - StarPU FAQs

for StarPU 1.4.5

Generated by Doxygen.

This manual documents the usage of StarPU version 1.4.5. Its contents was last updated on 2024-04-19.

Chapter 1

Organization

This part explains how to better tune your application to achieve good performance, and also how to fix some difficulties you may encounter while implementing your applications.

- We give a list of features in Chapter Check List When Performance Are Not There which should be checked to improve performances of your applications.
- There are some frequently asked questions in Chapter Frequently Asked Questions that may help you to solve your problems.

If you have problems that cannot be solved, please contact us.

Chapter 2

Check List When Performance Are Not There

TODO: improve!
To achieve good performance, we give below a list of features which should be checked.
For a start, you can use OfflinePerformanceTools to get a Gantt chart which will show roughly where time is spent, and focus correspondingly.

2.1 Check Task Size

Make sure that your tasks are not too small, as the StarPU runtime overhead may not be negligible. As explained in TaskSizeOverhead, you can run the script `tasks_size_overhead.sh` to get an idea of the scalability of tasks depending on their duration (in µs), on your own system.
Typically, 10µs-ish tasks are definitely too small, the CUDA overhead itself is much bigger than this.
1ms-ish tasks may be a good start, but will not necessarily scale to many dozens of cores, so it's better to try to get 10ms-ish tasks.
It may be useful to dedicate a whole core to the main thread, so it can spend its time on submitting tasks, by setting the STARPU_MAIN_THREAD_BIND environment variable to 1.
Tasks durations can easily be observed when performance models are defined (see PerformanceModelExample) by using the tools `starpu_perfmodel_plot` or `starpu_perfmodel_display` (see PerformanceOf↩ Codelets)
When using parallel tasks, the problem is even worse since StarPU has to synchronize the tasks execution.

2.2 Configuration Which May Improve Performance

If you do not plan to use support for GPUs or out-of-core, i.e. not use StarPU's ability to manage data coherency between several memory nodes, the `configure` option --enable-maxnodes=1 allows to considerably reduce Star↩ PU's memory management overhead.
The `configure` option --enable-fast disables all assertions. This makes StarPU more performant for tiny tasks by disabling all sanity checks. Only use this for measurements and production, not for development, since this will drop all basic checks.

2.3 Data Related Features Which May Improve Performance

As can be seen in StatesInGantt, if the application has a lot of different kinds of sizes of data, StarPU will end up freeing/reallocating data on GPU to accomodate for the different sizes. It can be very effective to round the allocated size up a bit by e.g. 10% (e.g. 11MB for all data sizes between 10MB and 11MB) so that StarPU will be able to reuse buffers of the same size for data with similar but not exactly same size. This can be registered by using starpu_matrix_data_register_allocsize(), starpu_vector_data_register_allocsize() so that StarPU records both the rounded-up data size, and the actual size used for computation.
link to DataManagement
link to DataPrefetch

2.4 Task Related Features Which May Improve Performance

link to TaskGranularity
link to TaskSubmission
link to TaskPriorities

2.5 Scheduling Related Features Which May Improve Performance

link to TaskSchedulingPolicy
link to TaskDistributionVsDataTransfer
link to Energy-basedScheduling
link to StaticScheduling

2.6 CUDA-specific Optimizations

For proper overlapping of asynchronous GPU data transfers, data has to be pinned by CUDA. Data allocated with starpu_malloc() is always properly pinned. If the application registers to StarPU some data which has not been allocated with starpu_malloc(), starpu_memory_pin() should be called to pin the data memory. Otherwise, the "Asynchronous copy submission" parts of the execution traces (see StatesInGantt) will show the synchronous inefficiency.
Note that CUDA pinning/unpinning takes a long time, so for e.g. temporary data, it is much more efficient to use a StarPU temporary data (see TemporaryData), that StarPU can reuse and thus avoid the pin/unpin cost.
Due to CUDA limitations, StarPU will have a hard time overlapping its own communications and the codelet computations if the application does not use a dedicated CUDA stream for its computations instead of the default stream, which synchronizes all operations of the GPU. The function starpu_cuda_get_local_stream() returns a stream which can be used by all CUDA codelet operations to avoid this issue. For instance:

```
func «<grid,block,0,starpu_cuda_get_local_stream()»> (foo, bar);
cudaError_t status = cudaGetLastError();
if (status != cudaSuccess) STARPU_CUDA_REPORT_ERROR(status);
cudaStreamSynchronize(starpu_cuda_get_local_stream());
```

as well as the use of `cudaMemcpyAsync()`, etc. for each CUDA operation one needs to use a version that takes a stream parameter.
If the kernel uses its own non-default stream, one can synchronize this stream with the StarPU-provided stream this way:

```
cudaEvent_t event;
call_kernel_with_its_own_stream()
cudaEventCreateWithFlags(&event, cudaEventDisableTiming);
cudaEventRecord(event, get_kernel_stream());
cudaStreamWaitEvent(starpu_cuda_get_local_stream(), event, 0);
cudaEventDestroy(event);
```

This code makes the StarPU-provided stream wait for a new event, which will be triggered by the completion of the kernel.
Unfortunately, some CUDA libraries do not have stream variants of kernels. This will seriously lower the potential for overlapping. If some CUDA calls are made without specifying this local stream, synchronization needs to be explicit with cudaDeviceSynchronize() around these calls, to make sure that they get properly synchronized with the calls using the local stream. Notably, `cudaMemcpy()` and `cudaMemset()` are actually asynchronous and need such explicit synchronization! Use `cudaMemcpyAsync()` and `cudaMemsetAsync()` instead.
Calling starpu_cublas_init() will ensure StarPU to properly call the CUBLAS library functions, and starpu_cublas_shutdown() will synchronously deinitialize the CUBLAS library on every CUDA device. Some libraries like Magma may however change the current stream of CUBLAS v1, one then has to call starpu_cublas_set_stream() at the beginning of the codelet to make sure that CUBLAS is really using the proper stream. When using CUBLAS v2, starpu_cublas_get_local_handle() can be called to queue CUBLAS kernels with the proper configuration.
Similarly, calling starpu_cusparse_init() makes StarPU create CUSPARSE handles on each CUDA device, starpu_cusparse_get_local_handle() can then be used to queue CUSPARSE kernels with the proper configuration. starpu_cusparse_shutdown() will synchronously deinitialize the CUSPARSE library on every CUDA device.
Similarly, calling starpu_cusolver_init() makes StarPU create CUSOLVER handles on each CUDA device, starpu_cusolverDn_get_local_handle(), starpu_cusolverSp_get_local_handle(), starpu_cusolverRf_get_local_handle(), can then be used to queue CUSOLVER kernels with the proper configuration. starpu_cusolver_shutdown() can be used to clear these handles. It is useful to use a STARPU_SCRATCH buffer whose size was set to the amount returned by `cusolver*Spotrf_bufferSize`. An example can be seen in `examples/cholesky`

If the kernel can be made to only use this local stream or other self-allocated streams, i.e. the whole kernel submission can be made asynchronous, then one should enable asynchronous execution of the kernel. This means setting the flag STARPU_CUDA_ASYNC in the corresponding field starpu_codelet::cuda_flags, and dropping the `cudaStreamSynchronize()` call at the end of the `cuda_func` function, so that it returns immediately after having queued the kernel to the local stream. That way, StarPU will be able to submit and complete data transfers while kernels are executing, instead of only at each kernel submission. The kernel just has to make sure that StarPU can use the local stream to synchronize with the kernel startup and completion.
Using the flag STARPU_CUDA_ASYNC also permits to enable concurrent kernel execution, on cards which support it (Kepler and later, notably). This is enabled by setting the environment variable STARPU_NWORKER_PER_CUDA to the number of kernels to be executed concurrently. This is useful when kernels are small and do not feed the whole GPU with threads to run.
Concerning memory allocation, you should really not use `cudaMalloc()` / `cudaFree()` within the kernel, since `cudaFree()` introduces way too many synchronizations within CUDA itself. You should instead add a parameter to the codelet with the STARPU_SCRATCH mode access. You can then pass to the task a handle registered with the desired size but with the `NULL` pointer, the handle can even be shared between tasks, StarPU will allocate per-task data on the fly before task execution, and reuse the allocated data between tasks.
See `examples/pi/pi_redux.c` for an example of use.

2.7 OpenCL-specific Optimizations

If the kernel can be made to only use the StarPU-provided command queue or other self-allocated queues, i.e. the whole kernel submission can be made asynchronous, then one should enable asynchronous execution of the kernel. This means setting the flag STARPU_OPENCL_ASYNC in the corresponding field starpu_codelet::opencl_flags and dropping the `clFinish()` and starpu_opencl_collect_stats() calls at the end of the kernel, so that it returns immediately after having queued the kernel to the provided queue. That way, StarPU will be able to submit and complete data transfers while kernels are executing, instead of only at each kernel submission. The kernel just has to make sure that StarPU can use the command queue it has provided to synchronize with the kernel startup and completion.

2.8 Detecting Stuck Conditions

It may happen that StarPU does not make progress for a long period of time. It may be due to contention inside StarPU, but it may also be an external problem, such as a stuck MPI or CUDA driver.
`export STARPU_WATCHDOG_TIMEOUT=10000` (STARPU_WATCHDOG_TIMEOUT)
allows making StarPU print an error message whenever StarPU does not terminate any task for 10ms, but lets the application continue normally. In addition to that,
`export STARPU_WATCHDOG_CRASH=1` (STARPU_WATCHDOG_CRASH)
raises `SIGABRT` in this condition, thus allowing to catch the situation in `gdb`.
It can also be useful to type `handle SIGABRT nopass` in `gdb` to be able to let the process continue, after inspecting the state of the process.

2.9 How to Limit Memory Used By StarPU And Cache Buffer Allocations

By default, StarPU makes sure to use at most 90% of the memory of GPU devices, moving data in and out of the device as appropriate, as well as using prefetch and write-back optimizations.
The environment variables STARPU_LIMIT_CUDA_MEM, STARPU_LIMIT_CUDA_devid_MEM, STARPU_LIMIT↩
_OPENCL_MEM, and STARPU_LIMIT_OPENCL_devid_MEM can be used to control how much (in MiB) of the GPU device memory should be used at most by StarPU (the default value is to use 90% of the available memory).
By default, the usage of the main memory is not limited, as the default mechanisms do not provide means to evict main memory when it gets too tight. This also means that by default, StarPU will not cache buffer allocations in main memory, since it does not know how much of the system memory it can afford.
The environment variable STARPU_LIMIT_CPU_MEM can be used to specify how much (in MiB) of the main memory should be used at most by StarPU for buffer allocations. This way, StarPU will be able to cache buffer allocations (which can be a real benefit if a lot of buffers are involved, or if allocation fragmentation can become a problem), and when using OutOfCore, StarPU will know when it should evict data out to the disk.

It should be noted that by default only buffer allocations automatically done by StarPU are accounted here, i.e. allocations performed through starpu_malloc_on_node() which are used by the data interfaces (matrix, vector, etc.). This does not include allocations performed by the application through e.g. malloc(). It does not include allocations performed through starpu_malloc() either, only allocations performed explicitly with the flag STARPU_MALLOC_COUNT, i.e. by calling

```
starpu_malloc_flags(STARPU_MALLOC_COUNT)
```

are taken into account. And starpu_free_flags() can be called to free the memory that was previously allocated with starpu_malloc_flags(). If the application wants to make StarPU aware of its own allocations, so that StarPU knows precisely how much data is allocated, and thus when to evict allocation caches or data out to the disk, starpu_memory_allocate() can be used to specify an amount of memory to be accounted for. starpu_memory_deallocate() can be used to account freed memory back. Those can for instance be used by data interfaces with dynamic data buffers: instead of using starpu_malloc_on_node(), they would dynamically allocate data with `malloc()/realloc()`, and notify StarPU of the delta by calling starpu_memory_allocate() and starpu_memory_deallocate(). By default, the memory management system uses a set of default flags for each node when allocating memory. starpu_malloc_on_node_set_default_flags() can be used to modify these default flags on a specific node.

starpu_memory_get_total() and starpu_memory_get_available() can be used to get an estimation of how much memory is available. starpu_memory_wait_available() can also be used to block until an amount of memory becomes available, but it may be preferable to call

```
starpu_memory_allocate(STARPU_MEMORY_WAIT)
```

to reserve this amount immediately.

2.10 How To Reduce The Memory Footprint Of Internal Data Structures

It is possible to reduce the memory footprint of the task and data internal structures of StarPU by describing the shape of your machine and/or your application when calling `configure`.

To reduce the memory footprint of the data internal structures of StarPU, one can set the `configure` parameters --enable-maxcpus, --enable-maxnumanodes, --enable-maxcudadev, --enable-maxopencldev and --enable-maxnodes to give StarPU the architecture of the machine it will run on, thus tuning the size of the structures to the machine.

To reduce the memory footprint of the task internal structures of StarPU, one can set the `configure` parameter --enable-maxbuffers to give StarPU the maximum number of buffers that a task can use during an execution. For example, in the Cholesky factorization (dense linear algebra application), the GEMM task uses up to 3 buffers, so it is possible to set the maximum number of task buffers to 3 to run a Cholesky factorization on StarPU.

The size of the various structures of StarPU can be printed by `tests/microbenchs/display_structures_size`.

It is also often useless to submit **all** the tasks at the same time. Task submission can be blocked when a reasonable given number of tasks have been submitted, by setting the environment variables STARPU_LIMIT_MIN_SUBMITTED_TASKS and STARPU_LIMIT_MAX_SUBMITTED_TASKS.

```
export STARPU_LIMIT_MAX_SUBMITTED_TASKS=10000
export STARPU_LIMIT_MIN_SUBMITTED_TASKS=9000
```

will make StarPU block submission when 10000 tasks are submitted, and unblock submission when only 9000 tasks are still submitted, i.e. 1000 tasks have completed among the 10000 which were submitted when submission was blocked. Of course this may reduce parallelism if the threshold is set too low. The precise balance depends on the application task graph.

These values can also be specified with the functions starpu_set_limit_min_submitted_tasks() and starpu_set_limit_max_submitted_ta

An idea of how much memory is used for tasks and data handles can be obtained by setting the environment variable STARPU_MAX_MEMORY_USE to `1`.

2.11 How To Reuse Memory

When your application needs to allocate more data than the available amount of memory usable by StarPU (given by starpu_memory_get_available()), the allocation cache system can reuse data buffers used by previously executed tasks. For this system to work with MPI tasks, you need to submit tasks progressively instead of as soon as possible, because in the case of MPI receives, the allocation cache check for reusing data buffers will be done at submission time, not at execution time.

There are two options to control the task submission flow. The first one is by controlling the number of submitted tasks during the whole execution. This can be done whether by setting the environment variables STARPU_LIMIT_MAX_SUBMITTED_TASKS and STARPU_LIMIT_MIN_SUBMITTED_TASKS to tell StarPU when to stop submit-

ting tasks and when to wake up and submit tasks again, or by explicitly calling starpu_task_wait_for_n_submitted() in your application code for finest grain control (for example, between two iterations of a submission loop).
The second option is to control the memory size of the allocation cache. This can be done in the application by using jointly starpu_memory_get_available() and starpu_memory_wait_available() to submit tasks only when there is enough memory space to allocate the data needed by the task, i.e. when enough data are available for reuse in the allocation cache.

2.12 Performance Model Calibration

Most schedulers are based on an estimation of codelet duration on each kind of processing unit. For this to be possible, the application programmer needs to configure a performance model for the codelets of the application (see PerformanceModelExample for instance). History-based performance models use on-line calibration. When using a scheduler which requires such performance model, StarPU will automatically calibrate codelets which have never been calibrated yet, and save the result in `$STARPU_HOME/.starpu/sampling/codelets`. The models are indexed by machine name. They can then be displayed various ways, see PerformanceOfCodelets .
By default, StarPU stores separate performance models according to the hostname of the system. To avoid having to calibrate performance models for each node of a homogeneous cluster for instance, the model can be shared by using `export STARPU_HOSTNAME=some_global_name` (STARPU_HOSTNAME), where `some_global_name` is the name of the cluster for instance, which thus overrides the hostname of the system.
By default, StarPU stores separate performance models for each GPU. To avoid having to calibrate performance models for each GPU of a homogeneous set of GPU devices for instance, the model can be shared by using the environment variables STARPU_PERF_MODEL_HOMOGENEOUS_CUDA, STARPU_PERF_MODEL_HOMOGENEOUS_OPENCL and STARPU_PERF_MODEL_HOMOGENEOUS_MPI_MS depending on your GPU device type.

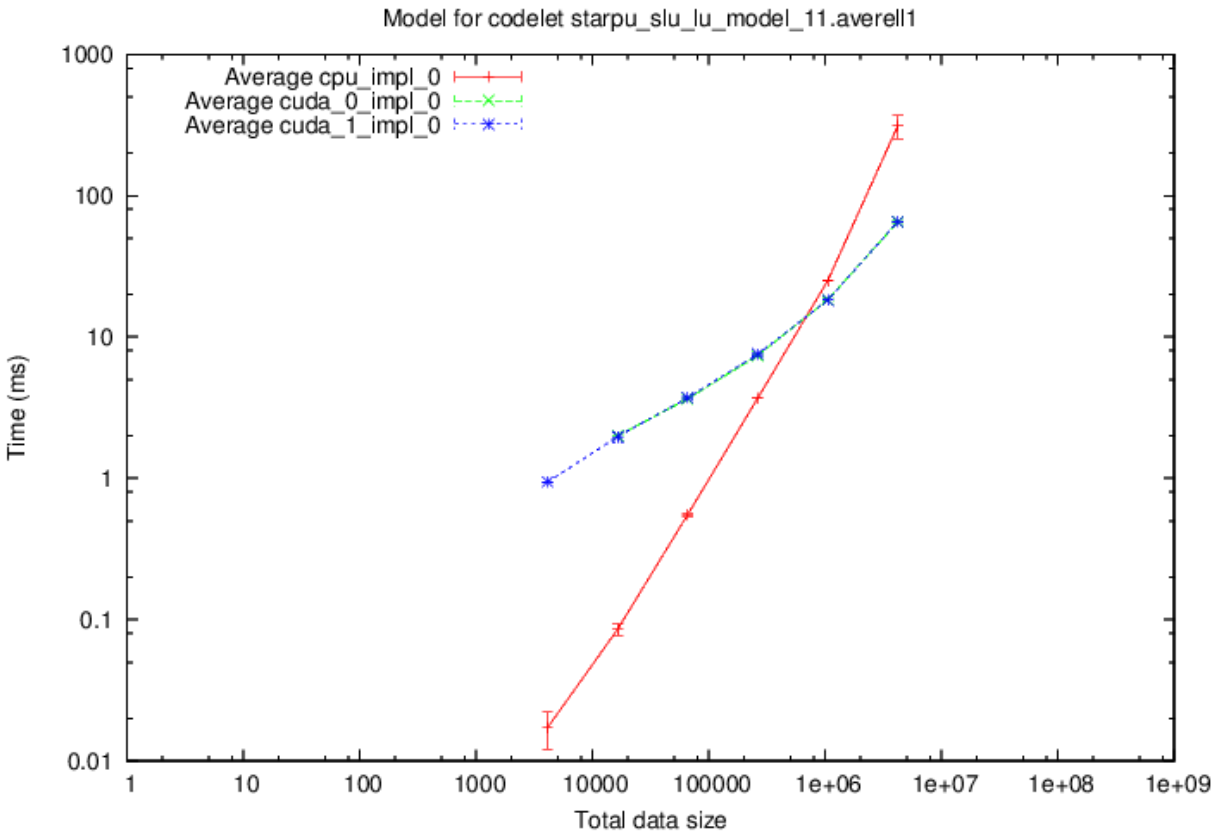
```
export STARPU_PERF_MODEL_HOMOGENEOUS_CUDA=1
export STARPU_PERF_MODEL_HOMOGENEOUS_OPENCL=1
export STARPU_PERF_MODEL_HOMOGENEOUS_MPI_MS=1
```

To force continuing calibration, use `export STARPU_CALIBRATE=1` (STARPU_CALIBRATE). This may be necessary if your application has not-so-stable performance. It may also be useful to use `STARPU_SCHED=eager` to get tasks distributed over the various workers. StarPU will force calibration (and thus ignore the current result) until 10 (`_STARPU_CALIBRATION_MINIMUM`) measurements have been made on each architecture, to avoid bad scheduling decisions just because the first measurements were not so good.
Note that StarPU will not record the very first measurement for a given codelet and a given size, because it would most often be hit by computation library loading or initialization. StarPU will also throw measurements away if it notices that after computing an average execution time, it notices that most subsequent tasks have an execution time largely outside the computed average ("Too big deviation for model..." warning messages). By looking at the details of the message and their reported measurements, it can highlight that your computation library really has non-stable measurements, which is probably an indication of an issue in the computation library, or the execution environment (e.g. rogue daemons).
Details on the current performance model status can be obtained with the tool `starpu_perfmodel_display`: the option `-l` lists the available performance models, and the option `-s` allows choosing the performance model to be displayed. The result looks like:

```
$ starpu_perfmodel_display -s starpu_slu_lu_model_getrf
performance model for cpu_impl_0
# hash    size     flops         mean          dev           n
914f3bef  1048576  0.000000e+00  2.503577e+04  1.982465e+02  8
3e921964  65536    0.000000e+00  5.527003e+02  1.848114e+01  7
e5a07e31  4096     0.000000e+00  1.717457e+01  5.190038e+00  14
...
```

It shows that for the LU 11 kernel with a 1MiB matrix, the average execution time on CPUs was about 25ms, with a 0.2ms standard deviation, over 8 samples. It is a good idea to check this before doing actual performance measurements.
A graph can be drawn by using the tool `starpu_perfmodel_plot`:

```
$ starpu_perfmodel_plot -s starpu_slu_lu_model_getrf
4096 16384 65536 262144 1048576 4194304
$ gnuplot starpu_starpu_slu_lu_model_getrf.gp
$ gv starpu_starpu_slu_lu_model_getrf.eps
```



If a kernel source code was modified (e.g. performance improvement), the calibration information is stale and should be dropped, to re-calibrate from start. This can be done by using `export STARPU_CALIBRATE=2` (STARPU_CALIBRATE).

Note: history-based performance models get calibrated only if a performance-model-based scheduler is chosen.

The history-based performance models can also be explicitly filled by the application without execution, if e.g. the application already has a series of measurements. This can be done by using starpu_perfmodel_update_history(), for instance:

```
static struct starpu_perfmodel perf_model =
{
    .type = STARPU_HISTORY_BASED,
    .symbol = "my_perfmodel",
};
struct starpu_codelet cl =
{
    .cuda_funcs = { cuda_func1, cuda_func2 },
    .nbuffers = 1,
    .modes = {STARPU_W},
    .model = &perf_model
};
void feed(void)
{
    struct my_measure *measure;
    struct starpu_task task;
    starpu_task_init(&task);
    task.cl = &cl;
    for (measure = &measures[0]; measure < measures[last]; measure++)
    {
        starpu_data_handle_t handle;
        starpu_vector_data_register(&handle, -1, 0, measure->size, sizeof(float));
        task.handles[0] = handle;
        starpu_perfmodel_update_history(&perf_model, &task, STARPU_CUDA_DEFAULT + measure->cudadev, 0,
      measure->implementation, measure->time);
        starpu_task_clean(&task);
        starpu_data_unregister(handle);
    }
}
```

Measurement has to be provided in milliseconds for the completion time models, and in Joules for the energy consumption models.

2.13 Profiling

A quick view of how many tasks each worker has executed can be obtained by setting `export STARPU_WORKER_STATS=1` (STARPU_WORKER_STATS). This is a convenient way to check that execution did happen on accelerators, without penalizing performance with the profiling overhead. The environment variable STARPU_WORKER_STATS_FILE can be defined to specify a filename in which to display statistics, by default statistics are printed on the standard error stream.
A quick view of how much data transfers have been issued can be obtained by setting `export STARPU_BUS_STATS=1` (STARPU_BUS_STATS). The environment variable STARPU_BUS_STATS_FILE can be defined to specify a filename in which to display statistics, by default statistics are printed on the standard error stream.
More detailed profiling information can be enabled by using `export STARPU_PROFILING=1` (STARPU_PROFILING) or by calling starpu_profiling_status_set() from the source code. Statistics on the execution can then be obtained by using `export STARPU_BUS_STATS=1` and `export STARPU_WORKER_STATS=1` . More details on performance feedback are provided in the next chapter.

2.14 Overhead Profiling

OfflinePerformanceTools can already provide an idea of to what extent and which part of StarPU brings an overhead on the execution time. To get a more precise analysis of which parts of StarPU bring the most overhead, `gprof` can be used.
First, recompile and reinstall StarPU with `gprof` support:

```
../configure --enable-perf-debug --disable-shared --disable-build-tests --disable-build-examples
```

Make sure not to leave a dynamic version of StarPU in the target path: remove any remaining `libstarpu-*.so`
Then relink your application with the static StarPU library, make sure that running `ldd` on your application does not mention any `libstarpu` (i.e. it's really statically-linked).

```
gcc test.c -o test $(pkg-config --cflags starpu-1.4) $(pkg-config --libs starpu-1.4)
```

Now you can run your application, this will create a file `gmon.out` in the current directory, it can be processed by running `gprof` on your application:

```
gprof ./test
```

This will dump an analysis of the time spent in StarPU functions.

Chapter 3

Frequently Asked Questions

3.1 How To Initialize A Computation Library Once For Each Worker?

Some libraries need to be initialized once for each concurrent instance that may run on the machine. For instance, a C++ computation class which is not thread-safe by itself, but for which several instantiated objects of that class can be used concurrently. This can be used in StarPU by initializing one such object per worker. For instance, the `libstarpufft` example does the following to be able to use FFTW on CPUs.
Some global array stores the instantiated objects:

```
fftw_plan plan_cpu[STARPU_NMAXWORKERS];
```

At initialization time of libstarpu, the objects are initialized:

```
int workerid;
for (workerid = 0; workerid < starpu_worker_get_count(); workerid++)
{
    switch (starpu_worker_get_type(workerid))
    {
        case STARPU_CPU_WORKER:
            plan_cpu[workerid] = fftw_plan(...);
            break;
    }
}
```

And in the codelet body, they are used:

```
static void fft(void *descr[], void *_args)
{
    int workerid = starpu_worker_get_id();
    fftw_plan plan = plan_cpu[workerid];
    ...
    fftw_execute(plan, ...);
}
```

We call starpu_worker_get_id() to retrieve the worker ID associated with the currently executing task, or call starpu_worker_get_id_check() with the error checking.
This however is not sufficient for FFT on CUDA: initialization has to be done from the workers themselves. This can be done thanks to starpu_execute_on_each_worker() or starpu_execute_on_each_worker_ex() with a specified task name, or starpu_execute_on_specific_workers() with specified workers. For instance, `libstarpufft` does the following.

```
static void fft_plan_gpu(void *args)
{
    plan plan = args;
    int n2 = plan->n2[0];
    int workerid = starpu_worker_get_id();
    cufftPlan1d(&plan->plans[workerid].plan_cuda, n, _CUFFT_C2C, 1);
    cufftSetStream(plan->plans[workerid].plan_cuda, starpu_cuda_get_local_stream());
}
void starpufft_plan(void)
{
    starpu_execute_on_each_worker(fft_plan_gpu, plan, STARPU_CUDA);
}
```

3.2 Hardware Topology

3.2.1 Interoperability hwloc

If hwloc is used, we can call starpu_get_hwloc_topology() to get the hwloc topology used by StarPU, and call starpu_get_pu_os_index() to get the OS index of a PU. We can call starpu_worker_get_hwloc_cpuset() to retrieve the hwloc CPU set associated with a worker.

3.2.2 Memory

There are various functions that we can use to retrieve information of memory node, such as to get the name of a memory node we call starpu_memory_node_get_name() and to get the kind of a memory node we call starpu_node_get_kind(). To retrieve the device ID associated with a memory node we call starpu_memory_node_get_devid(). We can call starpu_worker_get_local_memory_node() to retrieve the local memory node associated with the current worker. We can also specify a worker and call starpu_worker_get_memory_node() to retrieve the associated memory node. To get the type of memory node associated with a kind of worker we call starpu_worker_get_memory_node_kind(). If we want to know the total number of memory nodes in the system we can call starpu_memory_nodes_get_count(), and we can also retrieve the total number of memory nodes in the system that match a specific memory node kind by calling starpu_memory_nodes_get_count_by_kind(). We can call starpu_memory_node_get_ids_by_type() to get the identifiers of memory nodes in the system that match a specific memory node type. To obtain a bitmap representing logical indexes of NUMA nodes we can call starpu_get_memory_location_bitmap().

3.2.3 Workers

StarPU provides a range of functions for querying and managing the worker configurations on a given system. One such function is starpu_worker_get_count(), which returns the total number of workers in the system. In addition to this, there are also specific functions to obtain the number of workers associated with various processing units controlled by StarPU: to retrieve the number of CPUs we can call starpu_cpu_worker_get_count(), to retrieve the number of CUDA devices we can call starpu_cuda_worker_get_count(), to retrieve the number of HIP devices we can call starpu_hip_worker_get_count(), to retrieve the number of OpenCL devices we can call starpu_opencl_worker_get_count(), to retrieve the number of MPI Master Slave workers we can call starpu_mpi_ms_worker_get_count(), and to retrieve the number of TCPIP Master Slave workers we can call starpu_tcpip_ms_worker_get_count().
There are various functions that we can use to retrieve information of the worker. We call starpu_worker_get_name() to get the name of the worker, we call starpu_worker_get_devid() to get the device ID of the worker or call starpu_worker_get_devids() to retrieve the list of device IDs that are associated with a worker, and call starpu_worker_get_devnum() to get number of the device controlled by the worker which begin from 0. We call starpu_worker_get_subworkerid() to get the ID of sub-worker for the device. We call starpu_worker_get_sched_ctx_list() to retrieve a list of scheduling contexts that a worker is associated with. We call starpu_worker_get_stream_workerids() to retrieve the list of worker IDs that share the same stream as a given worker.
To retrieve the total number of NUMA nodes in the system we call starpu_memory_nodes_get_numa_count(). To get the device identifier associated with a specific NUMA node and to get the NUMA node identifier associated with a specific device we can call starpu_memory_nodes_numa_id_to_devid() and starpu_memory_nodes_numa_devid_to_id() respectively.
We can also print out information about the workers currently registered with StarPU. starpu_worker_display_all() prints out information of all workers, starpu_worker_display_names() prints out information of all the workers of the given type, starpu_worker_display_count() prints out the number of workers of the given type.
StarPU provides various functions associated to the type of processing unit, such as starpu_worker_get_type(), which returns the type of processing unit associated to the worker, e.g. CPU or CUDA. We can call starpu_worker_get_type_as_string() to retrieve a string representation of the type of a worker or call starpu_worker_get_type_from_string() to retrieve a worker type enumeration value from a string representation of a worker type or call starpu_worker_get_type_as_env_var() to retrieve a string representation of the type of a worker that can be used as an environment variable. Another function, starpu_worker_get_count_by_type(), returns the number of workers of a specific type. starpu_worker_get_ids_by_type() returns a list of worker IDs for a specific type, and starpu_worker_get_by_type() returns the ID of the specific worker that has the specific type, starpu_worker_get_by_devid() returns the ID of the worker that has the specific type and device ID. To get the type of worker associated with a kind of memory node we call starpu_memory_node_get_worker_archtype(). To check if type of processing unit matches one of StarPU's defined worker architectures we can call starpu_worker_archtype_is_valid(), while in order to convert an architecture mask to a worker architecture we can call starpu_arch_mask_to_worker_archtype().
To retrieve the binding ID of the worker associated with the currently executing task we can call starpu_worker_get_bindid(), it is useful for applications that require information about the binding of a particular task to a specific processor. We can call starpu_bindid_get_workerids() to retrieve the list of worker IDs that are bound to a given binding ID.
We can call starpu_workers_get_tree() to get information about the tree facilities provided by StarPU.

3.2.4 Bus

StarPU provides several functions to declare or retrieve information about the buses in a machine. The function starpu_bus_get_count() can be used to get the total number of buses available. To obtain the identifier of the bus between a source and destination point, the function starpu_bus_get_id() can be called. The source and destination points of a bus can be obtained by calling the functions starpu_bus_get_src() and starpu_bus_get_dst() respectively. Furthermore, users can use the function starpu_bus_set_direct() to declare that there is a direct link between a GPU and memory to the driver. The direct link can significantly reduce data transfer latency and improve overall performance. Moreover, users can use the function starpu_bus_get_direct() to retrieve information about whether a direct link has been established between a GPU and memory using the starpu_bus_set_direct() function. starpu_bus_set_ngpus() and starpu_bus_get_ngpus() functions can be used to declare and retrieve the number of GPUs of this bus that users need.

3.3 Using The Driver API

Running Drivers

```
int ret;
struct starpu_driver =
{
    .type = STARPU_CUDA_WORKER,
    .id.cuda_id = 0
};
ret = starpu_driver_init(&d);
if (ret != 0)
    error();
while (some_condition)
{
    ret = starpu_driver_run_once(&d);
    if (ret != 0)
        error();
}
ret = starpu_driver_deinit(&d);
if (ret != 0)
    error();
```

same as:

```
int ret;
struct starpu_driver =
{
    .type = STARPU_CUDA_WORKER,
    .id.cuda_id = 0
};
ret = starpu_driver_run(&d);
if (ret != 0)
    error();
```

The function starpu_driver_run() initializes the given driver, run it until starpu_drivers_request_termination() is called.
To add a new kind of device to the structure starpu_driver, one needs to:

1. Add a member to the union starpu_driver::id
2. Modify the internal function `_starpu_launch_drivers()` to make sure the driver is not always launched.
3. Modify the function starpu_driver_run() so that it can handle another kind of architecture. The function starpu_driver_run() is equal to call starpu_driver_init(), then to call starpu_driver_run_once() in a loop, and finally to call starpu_driver_deinit().
4. Write the new function `_starpu_run_foobar()` in the corresponding driver.

3.4 On-GPU Rendering

Graphical-oriented applications need to draw the result of their computations, typically on the very GPU where these happened. Technologies such as OpenGL/CUDA interoperability permit to let CUDA directly work on the OpenGL buffers, making them thus immediately ready for drawing, by mapping OpenGL buffer, textures or renderbuffer objects into CUDA. CUDA however imposes some technical constraints: peer memcpy has to be disabled, and the thread that runs OpenGL has to be the one that runs CUDA computations for that GPU.
To achieve this with StarPU, pass the option --disable-cuda-memcpy-peer to `configure` (TODO: make it dynamic), OpenGL/GLUT has to be initialized first, and the interoperability mode has to be enabled by using the

field starpu_conf::cuda_opengl_interoperability, and the driver loop has to be run by the application, by using the field starpu_conf::not_launched_drivers to prevent StarPU from running it in a separate thread, and by using starpu_driver_run() to run the loop. The examples `gl_interop` and `gl_interop_idle` show how it articulates in a simple case, where rendering is done in task callbacks. The former uses `glutMainLoopEvent` to make GLUT progress from the StarPU driver loop, while the latter uses `glutIdleFunc` to make StarPU progress from the GLUT main loop.
Then, to use an OpenGL buffer as a CUDA data, StarPU simply needs to be given the CUDA pointer at registration, for instance:

```
/* Get the CUDA worker id */
for (workerid = 0; workerid < starpu_worker_get_count(); workerid++)
        if (starpu_worker_get_type(workerid) == STARPU_CUDA_WORKER)
                break;
/* Build a CUDA pointer pointing at the OpenGL buffer */
cudaGraphicsResourceGetMappedPointer((void**)&output, &num_bytes, resource);
/* And register it to StarPU */
starpu_vector_data_register(&handle, starpu_worker_get_memory_node(workerid), output, num_bytes /
      sizeof(float4), sizeof(float4));
/* The handle can now be used as usual */
starpu_task_insert(&cl, STARPU_RW, handle, 0);
/* ... */
/* This gets back data into the OpenGL buffer */
starpu_data_unregister(handle);
```

and display it e.g. in the callback function.

3.5 Using StarPU With MKL 11 (Intel Composer XE 2013)

Some users had issues with MKL 11 and StarPU (versions 1.1rc1 and 1.0.5) on Linux with MKL, using 1 thread for MKL and doing all the parallelism using StarPU (no multithreaded tasks), setting the environment variable `MKL_NUM_THREADS` to `1`, and using the threaded MKL library, with `iomp5`.
Using this configuration, StarPU only uses 1 core, no matter the value of STARPU_NCPU. The problem is actually a thread pinning issue with MKL.
The solution is to set the environment variable KMP_AFFINITY to `disabled` (http://software.intel.com/sites/products/documentation/studio/composer/en-us/2011Update/compiler_c/optaps/common/optaps_openmp_thread_affinity.htm).

3.6 Thread Binding on NetBSD

When using StarPU on a NetBSD machine, if the topology discovery library `hwloc` is used, thread binding will fail. To prevent the problem, you should at least use the version 1.7 of `hwloc`, and also issue the following call:

```
$ sysctl -w security.models.extensions.user_set_cpu_affinity=1
```

Or add the following line in the file `/etc/sysctl.conf`

```
security.models.extensions.user_set_cpu_affinity=1
```

3.7 StarPU permanently eats 100% of all CPUs

Yes, this is on purpose.
By default, StarPU uses active polling on task queues to minimize wake-up latency for better overall performance. We can call starpu_is_paused() to check whether the task processing by workers has been paused or not.
If eating CPU time is a problem (e.g. application running on a desktop), pass option --enable-blocking-drivers to `configure`. This will add some overhead when putting CPU workers to sleep or waking them, but avoid eating 100% CPU permanently.

3.8 Interleaving StarPU and non-StarPU code

If your application only partially uses StarPU, and you do not want to call starpu_init() / starpu_shutdown() at the beginning/end of each section, StarPU workers will poll for work between the sections. To avoid this behavior, you can "pause" StarPU with the starpu_pause() function. This will prevent the StarPU workers from accepting new work (tasks that are already in progress will not be frozen), and stop them from polling for more work.

Note that this does not prevent you from submitting new tasks, but they won't execute until starpu_resume() is called. Also note that StarPU must not be paused when you call starpu_shutdown(), and that this function pair works in a push/pull manner, i.e. you need to match the number of calls to these functions to clear their effect.
One way to use these functions could be:

```
starpu_init(NULL);
starpu_worker_wait_for_initialisation(); // Wait for the worker to complete its initialization process
starpu_pause(); // To submit all the tasks without a single one executing
submit_some_tasks();
starpu_resume(); // The tasks start executing
starpu_task_wait_for_all();
starpu_pause(); // Stop the workers from polling
starpu_resume();
starpu_shutdown();
```

3.9 When running with CUDA or OpenCL devices, I am seeing less CPU cores

Yes, this is on purpose.
Since GPU devices are way faster than CPUs, StarPU needs to react quickly when a task is finished, to feed the GPU with another task (StarPU actually submits a couple of tasks in advance to pipeline this, but filling the pipeline still has to be happening often enough), and thus it has to dedicate threads for this, and this is a very CPU-consuming duty. StarPU thus dedicates one CPU core for driving each GPU by default.
Such dedication is also useful when a codelet is hybrid, i.e. while kernels are running on the GPU, the codelet can run some computation, which thus be run by the CPU core instead of driving the GPU.
One can choose to dedicate only one thread for all the CUDA devices by setting the STARPU_CUDA_THREAD_↩ PER_DEV environment variable to `1`. The application however should use STARPU_CUDA_ASYNC on its CUDA codelets (asynchronous execution), otherwise the execution of a synchronous CUDA codelet will monopolize the thread, and other CUDA devices will thus starve while it is executing.

3.10 StarPU does not see my CUDA device

First, make sure that CUDA is properly running outside StarPU: build and run the following program with `-lcudart`:

```
#include <stdio.h>
#include <cuda.h>
#include <cuda_runtime.h>
int main(void)
{
        int n, i, version;
        cudaError_t err;
        err = cudaGetDeviceCount(&n);
        if (err)
        {
                fprintf(stderr,"cuda error %d\n", err);
                exit(1);
        }
        cudaDriverGetVersion(&version);
        printf("driver version %d\n", version);
        cudaRuntimeGetVersion(&version);
        printf("runtime version %d\n", version);
        printf("\n");
        for (i = 0; i < n; i++)
        {
                struct cudaDeviceProp props;
                printf("CUDA%d\n", i);
                err = cudaGetDeviceProperties(&props, i);
                if (err)
                {
                        fprintf(stderr,"cudaGetDeviceProperties cuda error %d\n", err);
                        continue;
                }
                printf("%s\n", props.name);
                printf("%0.3f GB\n", (float) props.totalGlobalMem / (1«30));
                printf("%u MP\n", props.multiProcessorCount);
                printf("\n");
                err = cudaSetDevice(i);
                if (err)
                {
                        fprintf(stderr,"cudaSetDevice(%d) cuda error %d\n", err, i);
                        continue;
                }
                err = cudaFree(0);
                if (err)
```

```
                {
                        fprintf(stderr,"cudaFree(0) on %d cuda error %d\n", err, i);
                        continue;
                }
        }
        return 0;
}
```

If that program does not find your device, the problem is not at the StarPU level, but with the CUDA drivers, check the documentation of your CUDA setup. This program is available in the source directory of StarPU in `tools/gpus/check_cuda.c`, along with another CUDA program `tools/gpus/cuda_list.cu`.

3.11 StarPU does not see my OpenCL device

First, make sure that OpenCL is properly running outside StarPU: build and run the following program with `-lOpenCL`:

```
#include <CL/cl.h>
#include <stdio.h>
#include <assert.h>
int main(void)
{
    cl_device_id did[16];
    cl_int err;
    cl_platform_id pid, pids[16];
    cl_uint nbplat, nb;
    char buf[128];
    size_t size;
    int i, j;
    err = clGetPlatformIDs(sizeof(pids)/sizeof(pids[0]), pids, &nbplat);
    assert(err == CL_SUCCESS);
    printf("%u platforms\n", nbplat);
    for (j = 0; j < nbplat; j++)
    {
        pid = pids[j];
        printf("    platform %d\n", j);
        err = clGetPlatformInfo(pid, CL_PLATFORM_VERSION, sizeof(buf)-1, buf, &size);
        assert(err == CL_SUCCESS);
        buf[size] = 0;
        printf("        platform version %s\n", buf);
        err = clGetDeviceIDs(pid, CL_DEVICE_TYPE_ALL, sizeof(did)/sizeof(did[0]), did, &nb);
        if (err == CL_DEVICE_NOT_FOUND)
          nb = 0;
        else
          assert(err == CL_SUCCESS);
        printf("%d devices\n", nb);
        for (i = 0; i < nb; i++)
        {
            err = clGetDeviceInfo(did[i], CL_DEVICE_VERSION, sizeof(buf)-1, buf, &size);
            buf[size] = 0;
            printf("    device %d version %s\n", i, buf);
        }
    }
    return 0;
}
```

If that program does not find your device, the problem is not at the StarPU level, but with the OpenCL drivers, check the documentation of your OpenCL implementation. This program is available in the source directory of StarPU in `tools/gpus/check_opencl.c`.

3.12 There seems to be errors when copying to and from CUDA devices

You should first try to disable asynchronous copies between CUDA and CPU workers. You can either do that with the configuration parameter --disable-asynchronous-cuda-copy or with the environment variable STARPU_DISABLE_ASYNCHRONOUS_CUDA_COPY.

If your application keeps failing, you will find in the source directory of StarPU, a directory named `tools/gpus` with various programs. `cuda_copy.cu` is testing the direct or undirect copy between CUDA devices.

You can also try to just disable the direct gpu-gpu transfers (known to fail under some hardware/cuda combinations) by setting the STARPU_ENABLE_CUDA_GPU_GPU_DIRECT environment variable to 0.

3.13 I keep getting a "Incorrect performance model file" error

The performance model file, used by StarPU to record the performance of codelets, seem to have been corrupted. Perhaps a previous run of StarPU stopped abruptly, and thus could not save it properly. You can have a look at the

file if you can fix it, but the simplest way is to just remove the file and run again, StarPU will just have to re-perform calibration for the corresponding codelet.

Part I

Appendix

Chapter 4

The GNU Free Documentation License

Version 1.3, 3 November 2008

1. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of ``copyleft'', which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

2. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The ``Document'', below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as ``you''. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

   A ``Modified Version'' of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A ``Secondary Section'' is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

   The ``Invariant Sections'' are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The ``Cover Texts'' are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A ``Transparent'' copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not `Transparent'' is called`Opaque''.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The ``Title Page'' means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, ``Title Page'' means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The ``publisher'' means any person or entity that distributes copies of the Document to the public.

A section ``Entitled XYZ'' means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as `Acknowledgements'', Dedications'', `Endorsements'', or`History''.) To ``Preserve the Title'' of such a section when you modify the Document means that it remains a section ``Entitled XYZ'' according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

3. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

4. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a

computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

5. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

(a) Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

(b) List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

(c) State on the Title page the name of the publisher of the Modified Version, as the publisher.

(d) Preserve all the copyright notices of the Document.

(e) Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

(f) Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

(g) Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

(h) Include an unaltered copy of this License.

(i) Preserve the section Entitled ``History'', Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled ``History'' in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

(j) Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the ``History'' section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

(k) For any section Entitled `Acknowledgements'' or`Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

(l) Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

(m) Delete any section Entitled ``Endorsements''. Such a section may not be included in the Modified Version.

(n) Do not retitle any existing section to be Entitled ``Endorsements'' or to conflict in title with any Invariant Section.

(o) Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled ``Endorsements'', provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-↩ Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

6. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled ``History'' in the various original documents, forming one section Entitled `History''; likewise combine any sections EntitledAcknowledgements", and any sections Entitled `Dedications''. You must delete all sections EntitledEndorsements."

7. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

8. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an ``aggregate'' if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

9. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled `Acknowledgements'', Dedications", or ``History'', the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

10. TERMINATION

    You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

    However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

    Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

    Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

11. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License `‘or any later version’’ applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

12. RELICENSING

    `Massive Multiauthor Collaboration Site'' (orMMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A `Massive Multiauthor Collaboration'' (orMMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    `‘CC-BY-SA’’ means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    `‘Incorporate’’ means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is `‘eligible for relicensing’’ if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

4.1 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (C) *year your name*. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled `‘GNU Free Documentation License’’.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the `‘with...Texts.’’ line with this:

with the Invariant Sections being *list their titles*, with the Front-Cover Texts being *list*, and with the Back-Cover Texts being *list*.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.
If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.